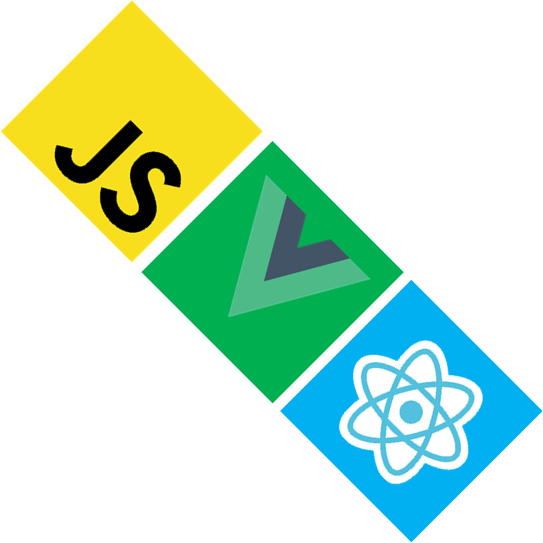
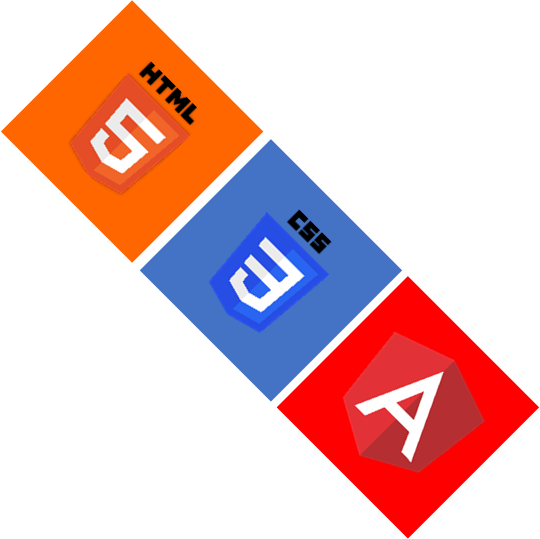
Verantwoording



Student: Jasper Wieringa

Leerlijn: React

Opleiding: HBO-ICT / Software Development

Product: CoffeeMachine

Datum: 14 juni 2020

Inhoudsopgave

Inleiding 3

Toegepaste technieken 4

Omschrijving aanpak 4

SweetCoffee.js 5

Model 5

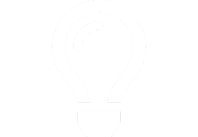
View 6

Controller 7

Tests 7

Gebruik Webapplicatie 8

Inleiding



Dit document wordt opgesteld naar aanleiding van de eindopdracht: React. Het doel van dit document is het verantwoorden van de gebouwde applicatie. In dit document wordt beschreven welke technieken er zijn toegepast en welke beweegredenen er waren om tot een bepaalde keuze te komen. Dit is opgesteld aan de hand van de beoordelingscriteria zoals hieronder omschreven:

* De student kan schone React code schrijven door o.a. principes toe te

passen die clean code dicteren.

* De student toont aan componenten en properties op de juiste manier toe

te kunnen passen.

* De student kan React componenten op de juiste manier voorzien van

interactie door middel van het toepassen van de daarbij horende React

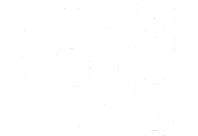
principes.

* De student kan aan de hand van het Data Context Interaction principe zorg

dragen voor ontkoppeling tussen React en de applicatieserver.

* De student kan test software toepassen om React code te testen.

Toegepaste technieken



Omschrijving aanpak

Om een zo’n schoon mogelijke React applicatie te maken, heb ik gekozen voor een opzet waarbij ik de componenten verdeel volgens onderstaand tabel.

|  |  |
| --- | --- |
| **Model :** | SweetCoffeeMock v.1.1.js  Status.js |
| **View/Window:** | Loader.js  Panel.js  Error.js |
| **Controller:** | CoffeeButton.js  Slider.js |



Om volledige controle over de styling te hebben en om de styling zoveel mogelijk binnen het component te houden, heb ik gebruik gemaakt van de volgende libraries:

* **Bootstrap**
* **Emotion**

SweetCoffee.js

Het hoofdcomponent die alle onderliggende componenten bevat. In dit component zitten slechts twee states, namelijk isError en een [errorDescr, errorMsg]. Ook kan hier de bugMultiplier worden ingesteld voor het willekeurig genereren van een bug a.h.v. de functie `generateRandomStatus()` in de **SweetCoffeeMock v.1.1.js** klasse. SweetCoffee.js parsed één callback functies mee, namelijk handleError en bevat de handleStatus functie.

* **handleError(status \_code)** wordt gebruikt om de isError state bij te werken om zo Panel.js of Error.js weer te geven.
* **handleStatus(status\_code)** wordt gebruikt om een status\_code op te vragen aan de **Status.js** klasse.

Model

Voor de Model(s) heb ik gekozen voor een simpele opzet.

**SweetCoffeeMock v.1.1.js**

Ik heb het ontvangen brondocument zo origineel mogelijk gelaten waarbij er een aantal extra functionaliteiten zijn toegevoegd, namelijk:

* **generateRandomStatus(bugMultiplier)** om een willekeurige foutmelding te kunnen genereren. De bugMultiplier (die is in te stellen in het SweetCoffee.js component) kan worden aangepast om meer (of minder) kans te hebben op een foutmelding.
* **Switch(name.toLowerCase())** een overkoepelende functie die a.h.v. een switch de juiste functie aanroept
* **handleStock()** callback vanuit het Panel.js component voor het updaten van de stock
* **checkStock()** om te controleren of de benodigde ingredienten nog aanwezig zijn in de huidige stock

**Status.js**

Deze klasse doet niets anders dan het returnen van een status object met daarin de omschrijving en het bericht.

View

Voor de View(s) heb ik gekozen voor een splitsing van taken met een Error component en een Panel component. Bij een fout, wordt het Error component weergegeven. Bij geen fout, kan zowel de Loader als de Panel worden geladen afhankelijk van de `isBusy` state.

**Error.js**

Voor deze functie heb ik gekozen voor een volledige container zoals in de bijgeleverde plaatjes het geval was. Dit component wordt aangeroepen door de isError variabele binnen de **SweetCoffeeMock v.1.1.js** klasse en wordt getoond op basis van 2 willekeurige getallen. Binnen dit component bevind zich een resetError() functie die wordt aangeroepen bij de onClick functie van de Reset knop om de machine te ‘resetten’.

**Panel.js**

Deze functie bevat een aantal states, namelijk stock, isBusy, busyWith en de sliderValues. Bij het renderen van dit component, wordt de stock opgehaald uit de **SweetCoffeeMock v.1.1.js** klasse. De isBusy state bepaalt of het Panel component, of het Loader component weergegeven dient te worden.

De busyWith state houdt bij welk drankje wordt bereidt.

De sliderValues state houdt bij welke waarden voor suiker- en melk er zijn geselecteerd.

Naast de states, bevat het Panel.js component een aantal functies, namelijk:

* **handleStock()** callback functie die wordt meegegeven aan de **SweetCoffeeMock v.1.1.js** klasse voor het updaten van de stock
* **handleSlidervalues()** voor het updaten van de state sliderValues.
* **handleDrink()** voor het bereiden van het drankje en de call naar de **SweetCoffeeMock v.1.1.js** klasse. In deze functie worden er een aantal states geupdate, namelijk isBusy, busyWith, handleStock (a.h.v. de callback) en sliderValues.
  1. **isBusy** wordt geupdate zodat het juiste component kan worden getoond
  2. **busyWith** wordt geupdate zodat het juiste drankje kan worden weergegeven in de Loader.js
  3. **handleStock** wordt geupdate omdat de Panel.js de state van stock bevat en deze geupdate dient te worden nadat elk drankje successvol is gemaakt.
  4. **sliderValues** wordt geupdate omdat na een keuze te hebben gemaakt in de slider, de waarde van count (de state van Slider.js) niet wordt bijgewerkt.

**Loader.js**

Deze functie doet vrijwel niets anders dan het tonen van een .gif en het weergeven van de props.drinkType in `Machine maakt een <<type drankje>> , even geduld...

Controller

Voor de Controller(s) heb ik gekozen voor twee componenten namelijk:

1. CoffeeButton.js (conflict met Bootstrap Button bij keuze Button.js)
2. Slider.js

**CoffeeButton.js**

Het CoffeeButton component returned slechts een button met een onClick functie die de callback functie props.handleDrink() aanroept en daar een aantal props aan meegeeft. De hoeveelheid buttons wordt bepaald a.h.v. JSON `drinkTypes.json ` die wordt ingeladen in het Panel.js component.

**Slider.js**

Het Slider component returned een Bootstrap slider en behoudt enkel de count state. Deze count wordt geupdate onChange van de slider. Terwijl deze count wordt geupdate, wordt ook de handleSliderValues() callback functie vanuit de Panel.js aangeroepen om te bepalen welke ingredienten er benodigd zijn. Er is een functie toegevoegd die een tooltip genereert om zo de selectie (in percentage) bij te houden.

Tests

Voor de tests heb ik gekozen om met name de functionaliteit van de **SweetCoffeeMock v.1.1.js** en de **Status.js** klassen te testen, omdat dat de componenten zijn die een return value geven, daarnaast is elk component getest of het correct wordt gerendered zonder foutmeldingen.

**Status.js**

Getest met vier verschillende tests of de returnvalue van het type gelijk is aan “object” is bij het correct invoeren van een status\_code tussen de 0-4.

Getest of de returnvalue van status.getStatusCode().message gelijk is aan “Onbekende fout” bij een foutieve invoer die groter dan, of gelijk is aan 4.

**SweetCoffeeMock v.1.1.js**

Getest of de returnvalue gelijk is aan een “boolean” (dit kan true of false zijn op basis van de generateRandomStatus() functie) bij het invoeren van correcte waarden

Getest of de returnvalue gelijk is aan false bij een foutief (niet bestaand) drankje

Getest of de returnvalue gelijk is aan false bij een lege voorraad

Getest of de returnvalue gelijk is aan false bij het doorvoeren van een “string” ipv “object” voor de`stock` state.

Getest of de returnvalue gelijk is aan een “boolean” (dit kan true of false zijn op basis van de generateRandomStatus() functie) bij het invoeren van ingredienten van het type “string” ipv “Number”. Deze worden door de **SweetCoffeeMock v.1.1** klasse omgezet naar een Number

Gebruik Webapplicatie

Run npm install

Run npm start om de applicatie te starten

Run npm test om alle tests uit te voeren